Design documentation

Use Cases

Scenario 1 - Register:

- **Description:** To enjoy JumpPack's features, the user must create an account in our database so that he/she can get their own unique experience.
- **Pre-Condition:** The username must be unique and with no special characters, the password must be at least 6 characters while having at least one uppercase and one special character or number (`!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~"`).
- **Post-Condition:** Insurance that all the pre-conditions are satisfied. Encryption of the password and upload all the input to the database.
- **Data:**
    - **Input:**
        - Username;
        - E-mail;
        - Password.
    - **Output:**
        - Feedback for username, e-mail and password;
        - Request verification;
        - Save account to database;
        - Load welcome page.
- **Interaction Steps**:
    - **Valid Case:**
        - Username is unique without special characters;
        - The password has at least 6 characters, one special character or number and one uppercase character;
        - The e-mail address is valid;
        - If everything is completed correctly the text boxes will appear with a green highlight and the account can be created.
    - **Alternative case:** The username has already been taken and/or the password does not mean the specified requirements and will therefore highlight the incorrect text field and give an appropriate message to assist the user
    - **Error Case:** If any of the valid cases are not satisfied the form where the case(s) is/are invalid appear with a red background and an error message is displayed telling the user how this form should be submitted.

Scenario 2 - Log In:

- **Description:** In order to access JumpPack system and to authenticate the user, the user must enter his username and password previously created and verified so he can have access to it's features.
- **Pre-Condition:** The username and password are correct and it satisfies the data in the database;
- **Post-Condition:** Insurance that all the pre-conditions are satisfied. A welcome page is presented to the user.
- **Data:**
    - **Input:**
        - Username;
        - Password.
    - **Output:**
        - Feedback for username and password;
        - Load welcome page.
- **Interaction Steps:**
    - **Valid case:** The username and password match the data in the database => A message is displayed saying that the user was successfully authenticated.
    - **Alternative case:** The user knows his username but forgot the password. After a few attempts a button is displayed with a message "Did you forget your password?" giving the opportunity to the user to get a new generated password sent to his e-mail so he can recover his account.
    - **Error case:** If the details are entered wrong or they don't match the database data a message is displayed saying that these details are wrong.

Scenario 3 - Browse Store:

- **Description:** The user has a wide variety of games that can obtain, including details of the same as the title, genre, requirements to play it, etc.
- **Pre-condition:** The user has logged in successfully.
- **Post-condition:** Insurance that all the pre-conditions are satisfied. A page with all the games and their details is presented.
- **Data:**
    - **Input:**
        - The user is able to filter the games he wants to see by their genre;
        - The user is able to obtain a game.
    - **Output:**
        - Recommended games against what the user previously played;
        - Games displayed by the filter chosen by the user;
        - Obtained game is added to the user's library.
- **Interaction Steps:**
    - **Valid case:**

        - The user is logged in.
        - =>
        - All the games are displayed;
        - Obtained games go to the library.
    - **Alternative case:** None.
    - **Error case:** The user is not logged in or lost his access to the internet => A message is displayed saying that currently the games can not be displayed or the chosen game cannot be obtained because the user might have lost connection to the internet.

Scenario 4 - Browse Library:

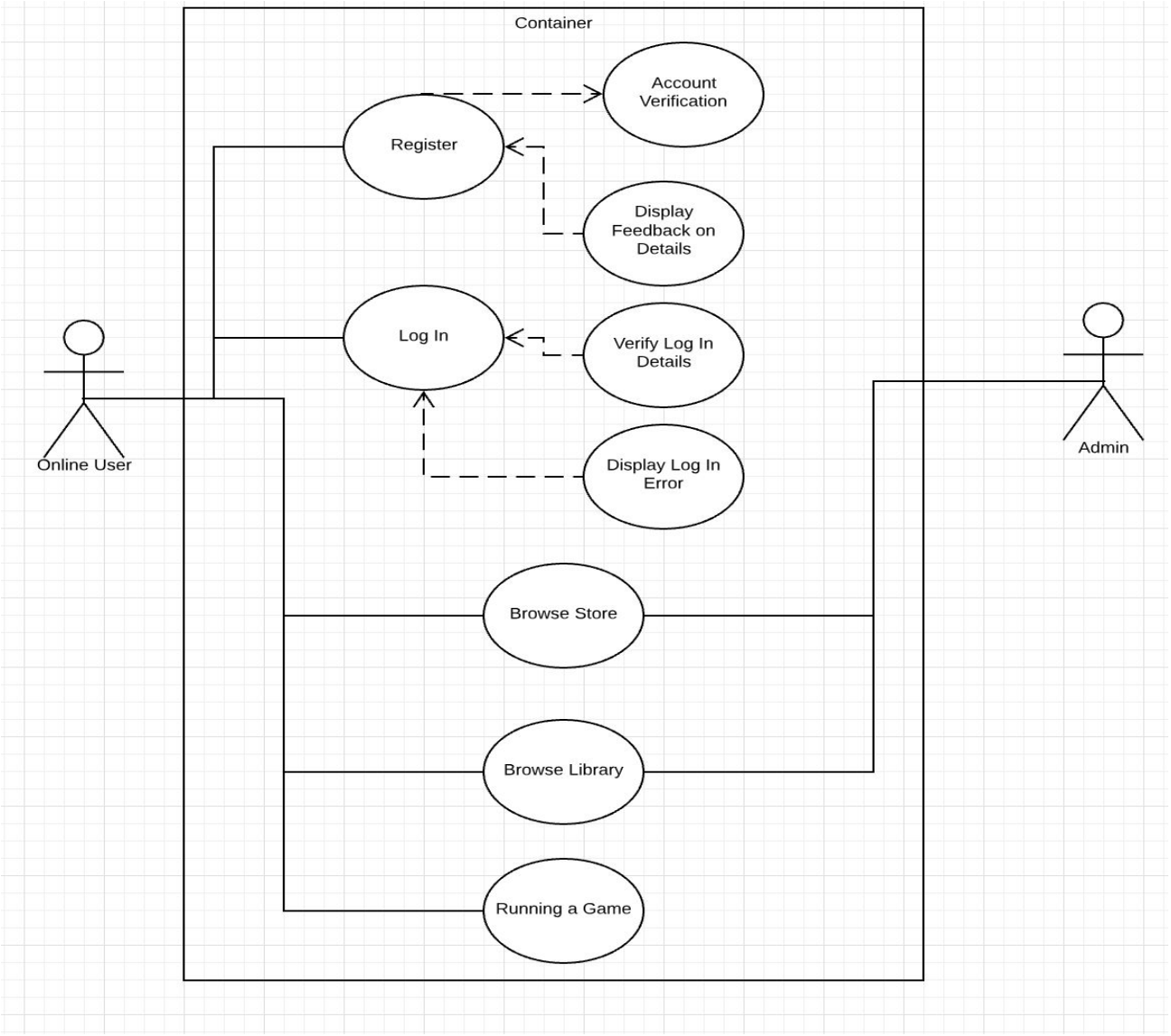
- **Description:** The user has a library with all the games obtained from the store or manually added. The user can launch the games from this library or install obtained games from the store.
- **Pre-condition:** The user has logged in successfully.
- **Post-condition:** Insurance that all the pre-conditions are satisfied. A page with all the games owned and their details is presented.
- **Data:**
    - **Input:** The user is able to manually add a game to the library.
    - **Output:** Games obtained in the store or added manually as well as information about them and time played.
- **Interaction Steps:**
    - **Valid case:** The user is logged in => All the games are displayed.
    - **Alternative case:** None.
    - **Error case:**
        - The user is not logged in or lost his access to the internet => A message is displayed saying that currently the games can not be displayed or the chosen game cannot be added to the library because the user might have lost connection to the internet or the account might be logged off.

Scenario 5 -Opening/Running a game:

- **Description:** The user select a game from the library to run it and play.
- **Pre-condition:** The user has this game in his library and this game is installed in his machine.
- **Post-condition:** Insurance that all the pre-conditions are satisfied. The game is runned so the user can play it.
- **Data:**
    - **Input:** The user chooses the game he wants to play.
    - **Output:** The game is opened and start running.
- **Interaction Steps:**

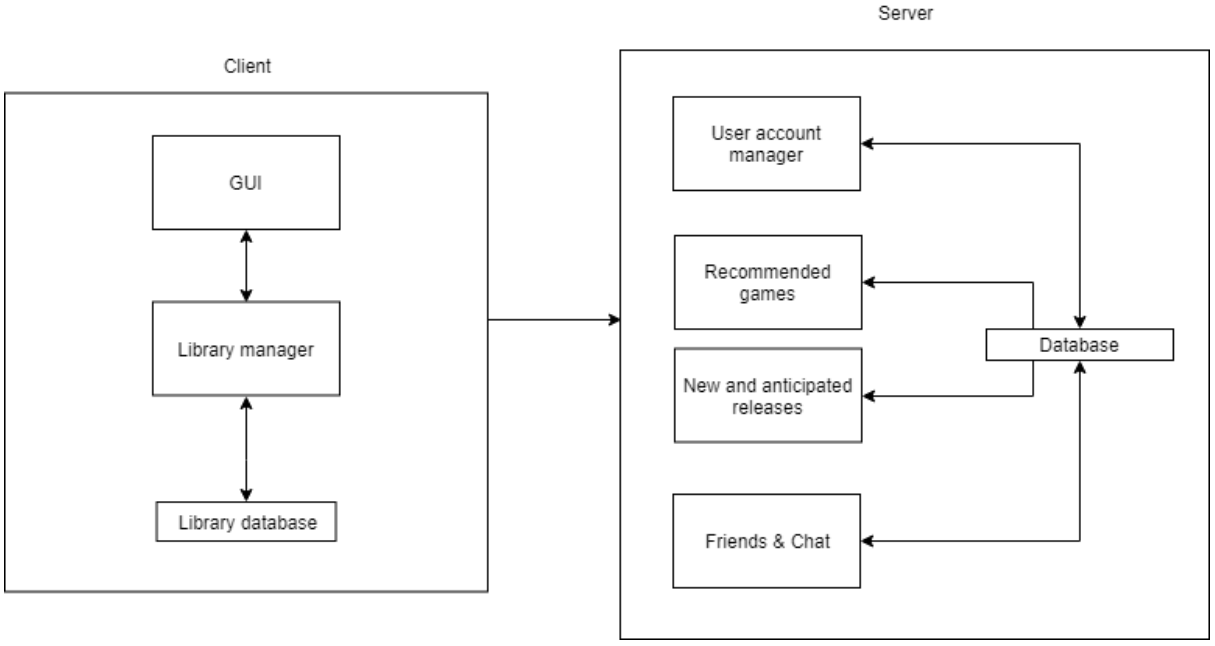
- **Valid case:** The user has the game installed and ready to run => Game runs.
- **Alternative case**: None.
- **Error case:**
  - The user has not the game installed in his machine;
  - The user is not logged in or lost his access to the internet.
  - **=>**
  - A message is displayed saying the user must have the game installed before run it or other message is displayed saying that the game can not be run because the user might have lost connection to the internet or the account might be logged off.

Use Case Diagram



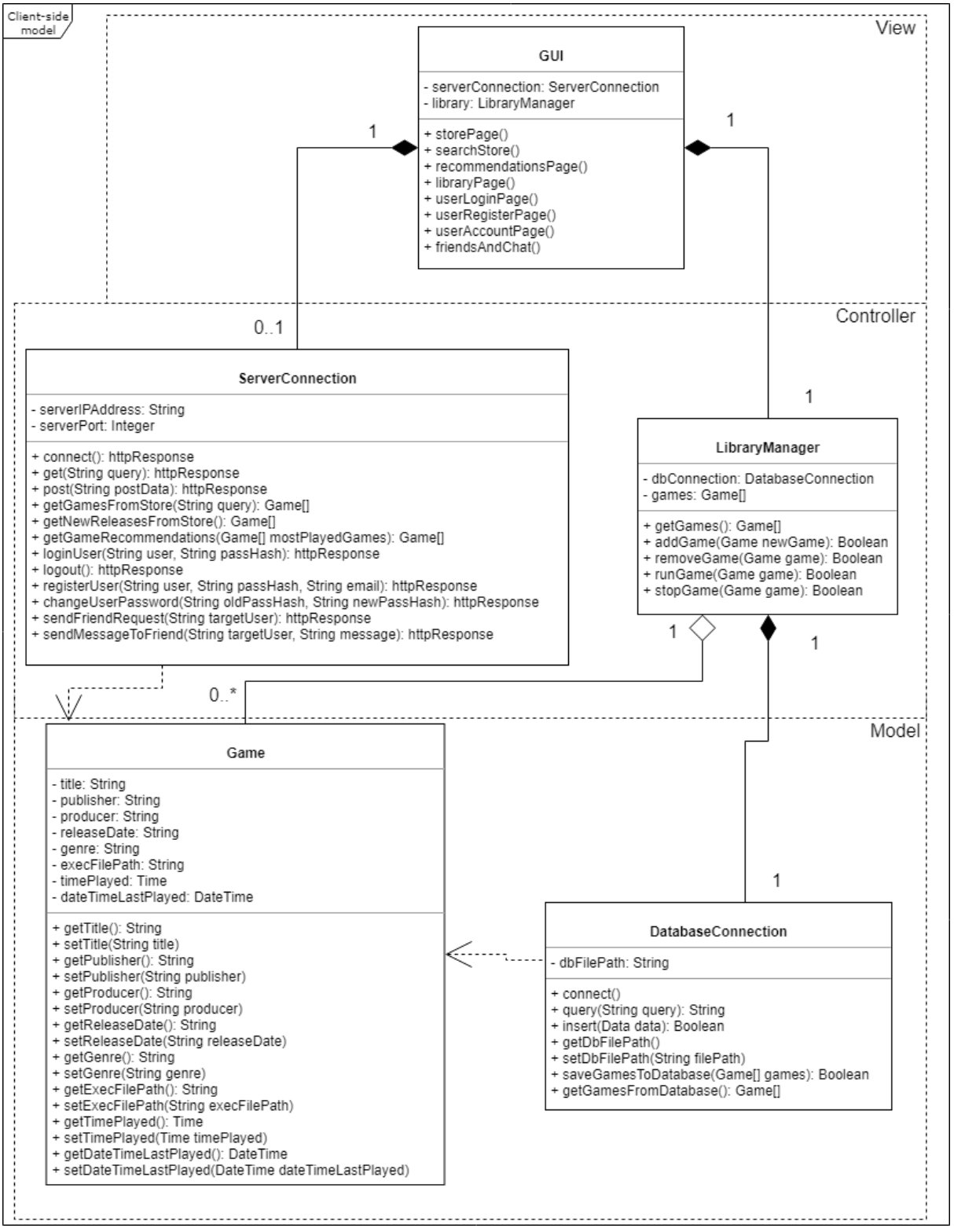
System Architecture

We used the client-server architectural pattern because it is object oriented and also helps to meet the non-functional requirements of the system. The first and most important requirement is security and the client-server pattern suits best because each client in the model does not need to know about one another. Secondly the system must be able to run on any operating system that supports JVM and since each client has the same general functionality, nothing needs to be replicated and the server can provide services which the client can request. The system must be easily modifiable and because the client-server model is split up into components we therefore have less dependencies and less chance of breaking the system as a result of new additions. Performance requirements may depend on the network which may be unpredictable however the client can still provide the user with basic offline services such as managing their game library.



From this top-down system architecture model the client provides the main service ‘library manager’ which allows the user to add/delete/edit and run games to their library without being connected to the server. The server provides four services to the user’s client, firstly ‘user account manager’ provides the user login, logout and registration and change password functionality, secondly ‘friends and chat’ service allows authorized users to add, remove and chat with their friends. The recommended games service will send recommended games data to the client based on usage from the user’s game library database. The system provides the client with new and anticipated releases data for the client to show in the GUI. Performance of the store page is highly dependent on network connectivity.

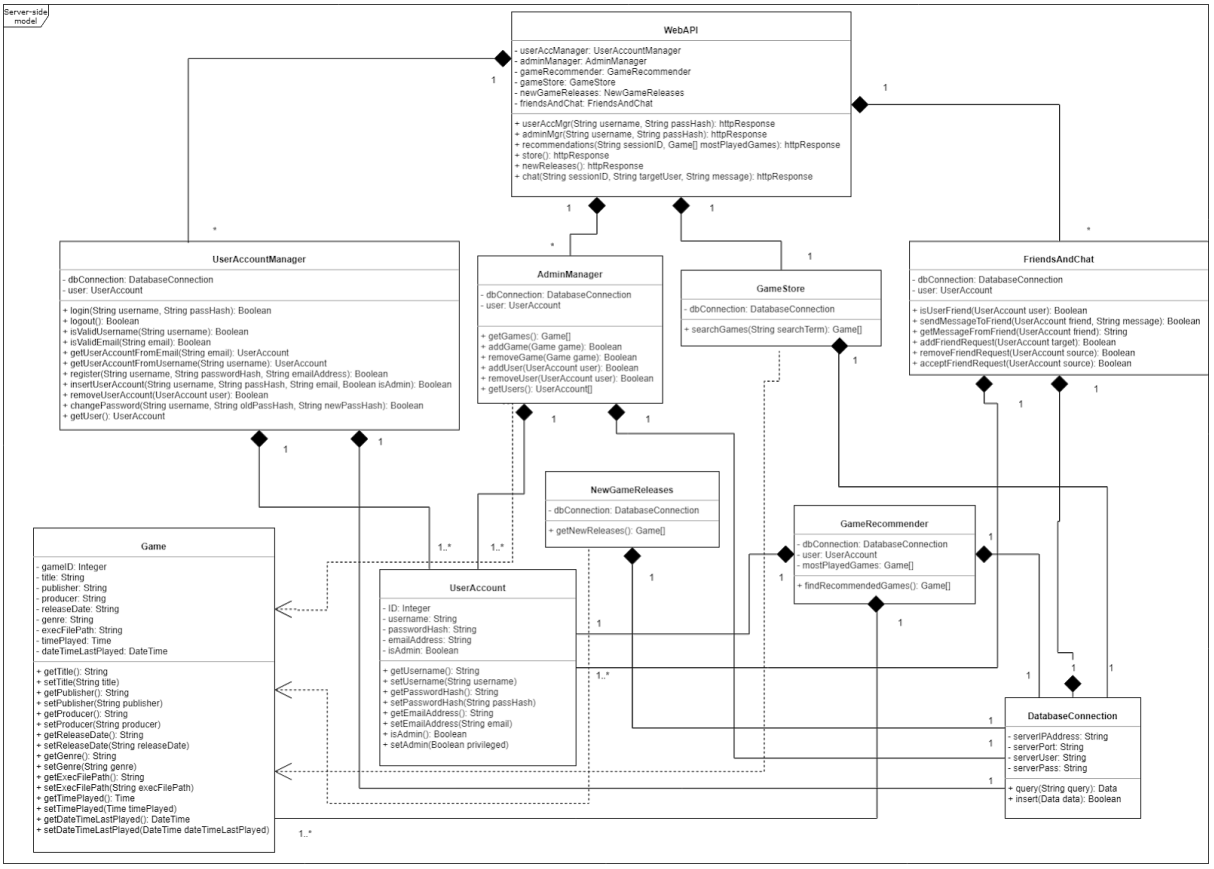
Client-side model



During the design of the client-side I noticed that the model also fits with the MVC architecture pattern, where the Game and Database classes are models and the GUI is the view and the controllers manipulate the data in the model to show in the view. This simplifies

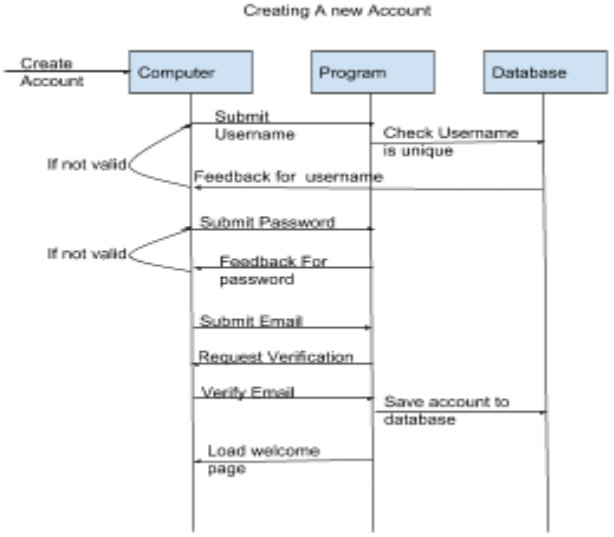
the system by breaking it down into layers as well as components. The advantages of this pattern is that components are loosely coupled (less dependencies) and highly cohesive.

Server-side model

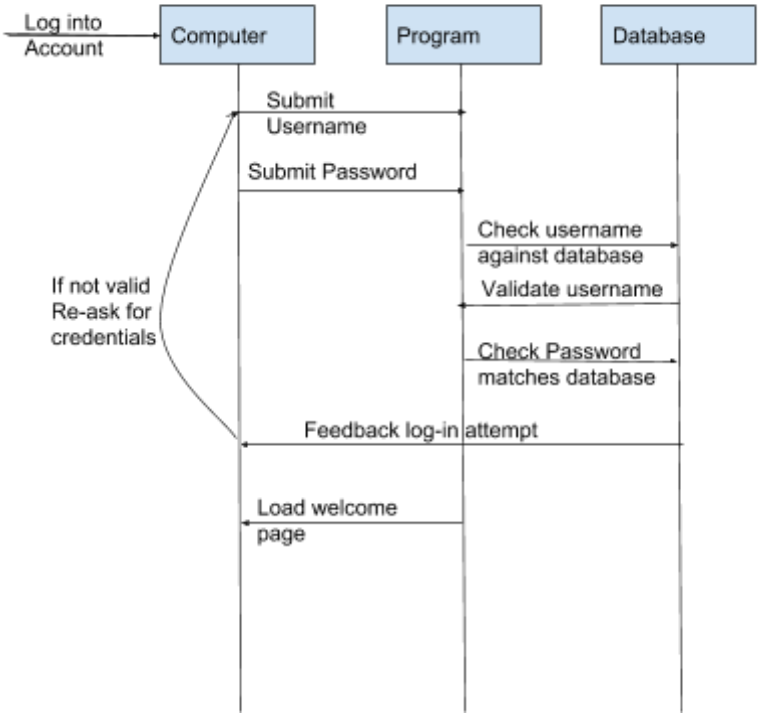


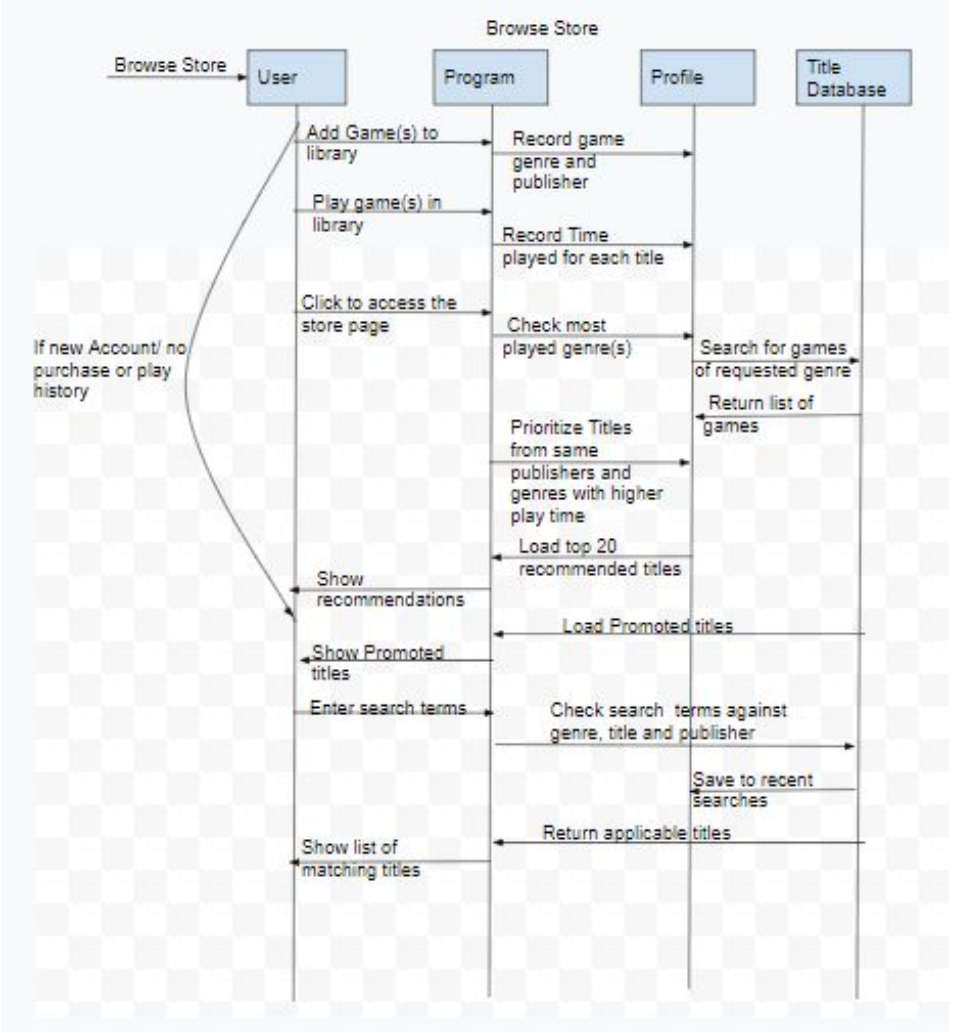
This also fits in with the MVC / client-server architecture pattern. The game class is reused from the client-side so that we do not need to replicate.

System Sequence Diagrams

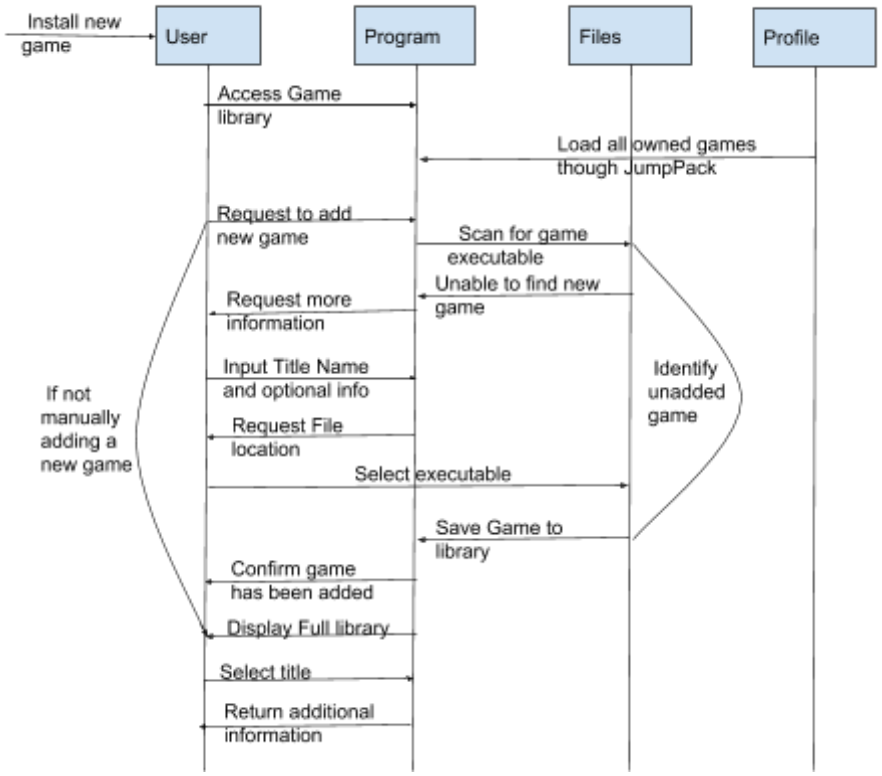


Logging into JumpPack

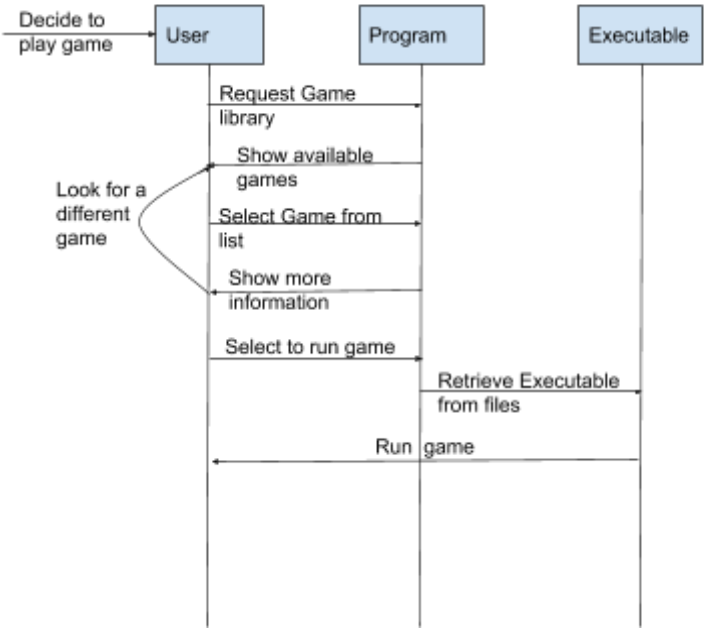




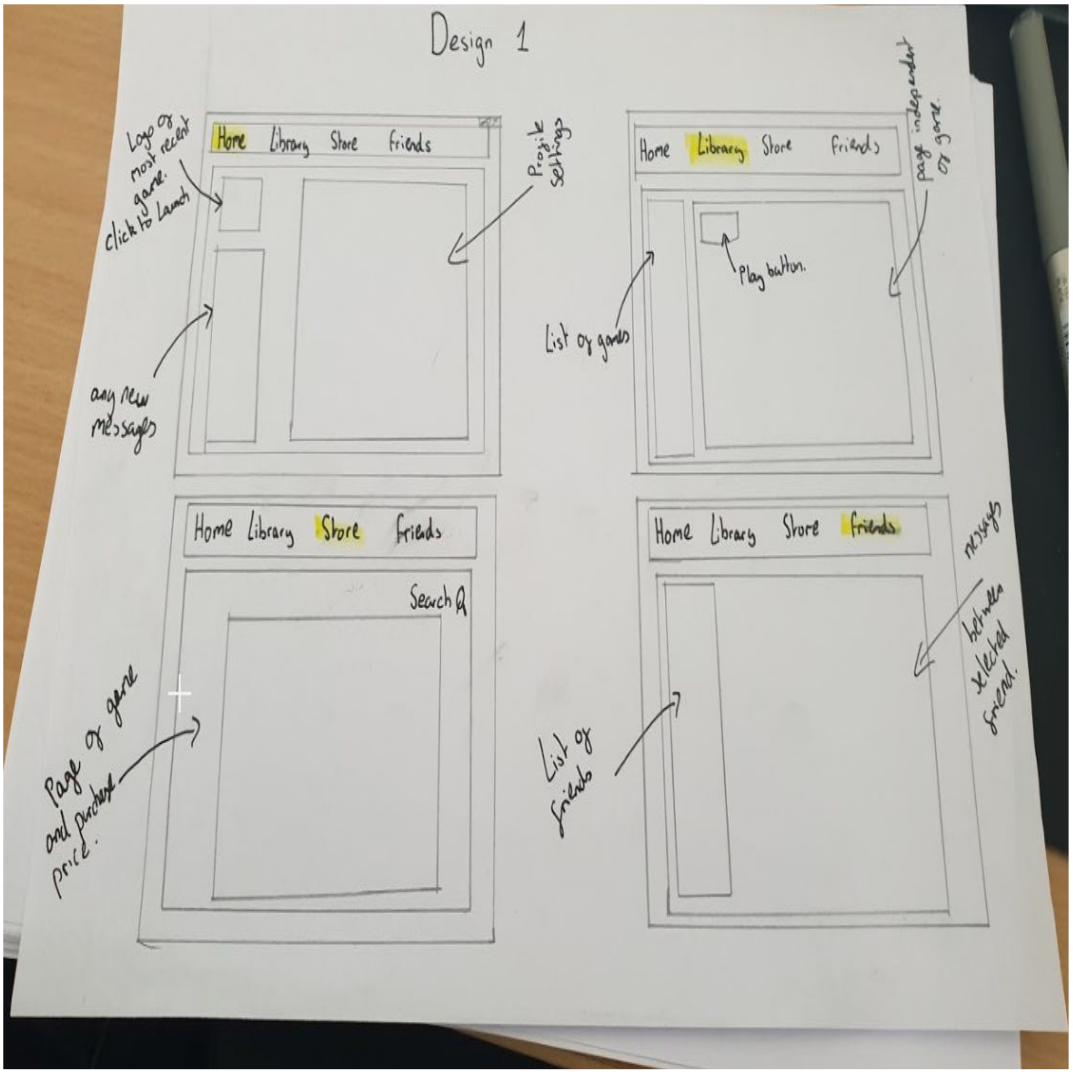
Browsing library/ Adding new games



Opening/Running A Game



System UI



As you can see there are 4 different screens that will be part of our system. On the homepage there will be a logo of the most recent game that the user has launched. And any of the new messages you may have received from friends. As well as that there will be a page where you can customise your profile such as the name and the picture associated.

On the Library page there will be a list of all the games owned that you can click and the rest of the page will have a unique image and relevant news to that game as well as play button to launch the game from.

In the store page there will be a featured games which have been selected based off games you already own and you can search for games via title or genre.

On the friends page there will be a list of all your friends plus a way you can send them messages and see previous messages.